Bash Prompt HOWTO

Giles Orr, *giles@interlog.com*, Vertaald door: Ellen Bokhorst, bokkie@nl.linux.org Revisie: 0.76
31/12/1999 23:20:55 giles

Het aanmaken en beheren van terminal en xterm-prompts wordt besproken, waaronder de standaard escape sequences om een gebruikersnaam, de huidige werkdirectory, tijd, enz. op te nemen. Verder worden er aanbevelingen gedaan over het aanpassen van de titelbalken van een xterm, het gebruiken van externe functies om te voorzien in informatie voor de prompt, en hoe je ANSI-kleuren kunt gebruiken.

Inhoudsopgave

1 Introductie en Administratieve zaken

1.1 Benodigdheden

Je hebt Bash nodig. De standaardversie voor de meeste Linux-distributies is óf 1.14.7, óf 2.0.x. Versie 1.14.7 was jarenlang de standaard, maar deze wordt langzamerhand vervangen. Ik heb Bash 2.0.x nu al een poosje gebruikt, maar bijna alle code die hier gepresenteerd wordt zou ook onder 1.14.7 moeten werken. Als ik me bewust ben van een probleem, zal ik het vermelden. Je kunt jouw Bash-versie controleren door achter de prompt `echo $BASH_VERSION` in te typen. Op mijn computer antwoordt het met `2.03.6(1)-release`.

Ervaring in het programmeren met de shell zou fijn zijn, maar dit is niet essentieel. Hoe meer je weet, hoe complexer de prompts die je zult kunnen maken. In de loop van deze tutorial ga ik uit van een basiskennis in het programmeren van de shell en van de Unix-utility's. Mijn eigen ervaringen in het programmeren van de shell zijn echter beperkt, dus ik geef veel voorbeelden en uitleg, die voor een ervaren shell-programmeur onnodig kunnen blijken.

1.2 Hoe dit document te gebruiken

Ik voeg heel veel voorbeelden en uitleg toe. Diverse onderdelen zullen voor verschillende mensen van variërende bruikbaarheid zijn. Het is zo'n lang document geworden dat het niet te doen is om het aan één stuk door te lezen. Lees gewoon die secties die je nodig hebt en ga zonodig naar een vorig deel.

1.3 Vertalingen

Japans: *http://www.jf.linux.or.jp/JF/JF-ftp/other-formats/Bash-Prompt/Bash-Prompt-HOWTO.html*, vertaald door Akira Endo, *akendo@t3.rim.or.jp*.

Duits: aan een vertaling wordt gewerkt door Thomas Keil, *thomas@h-preissler.de*.

Italiaans: door Daniel Dui, *ddui@iee.org*, beschikbaar op *http://www.crs4.it/~dui/linux.html*.

Portugees: vertaling in bewerking door Mário Gamito, *mario.gamito@mail.telepac.pt*.

Spaans: aan een vertaling wordt gewerkt door Iosu Santurtún *iosu@bigfoot.com* op *http://mipagina.euskaltel.es/iosus/linux/Bash-Prompt-HOWTO.html*.

Nederlands: aan een vertaling wordt gewerkt door Ellen Bokhorst *bokkie@nl.linux.org*, en het zal beschikbaar zijn op *http://www.nl.linux.org/doc/HOWTO*.

Chinees: aan een vertaling wordt gewerkt door Allen Huang *lancelot@tomail.com.tw*. Ik zal een URL opnemen wanneer ik het heb.

Veel dank aan iedereen! URL's zullen worden toegevoegd zodra ze beschikbaar zijn.

Als je aan een vertaling werkt, laat me dit dan alsjeblieft weten, vooral als het beschikbaar is op een te linken URL. Bedankt.

1.4 Problemen

Dit is een lijst met problemen die ik tegen kwam tijdens het programmeren van prompts. Begin niet hier te lezen, en laat deze lijst je niet ontmoedigen. Het zijn voornamelijk zeer kleine details. Controleer het gewoon als je iets raars tegenkomt.

- Veel van de mogelijkheden van Bash (zoals onder andere math binnen $(())) zijn opties die tijdens het compileren kunnen worden opgegeven. Als je gebruik maakt van een binaire distributie zoals die met een Linux-distributie wordt meegeleverd, zouden al dergelijke mogelijkheden moeten zijn meegecompileerd. Maar als je op het systeem van iemand anders werkt, is het waard om dit in gedachten te houden, wanneer iets niet werkt zoals je het zou verwachten. Een aantal opmerkingen hierover staan in *Learning the Bash Shell*, p.260-262.
- De terminal screen manager "screen"handelt ANSI-kleuren niet altijd goed af. Ik ben helaas geen screen-expert. Mijn huidige versie van screen (3.7.6-1, een RPM package) schijnt in alle situaties goed te werken, maar ik heb situaties gezien waar screen alle kleuren van de prompt in X-terminals terugbracht naar de standaard-voorgrondkleur. Dit schijnt onder de console geen probleem te zijn.
- Xdefaults bestanden kunnen kleuren overschrijven. Kijk in ˜/.Xdefaults voor regels die refereren naar XTerm*background en XTerm*foreground (of mogelijk XTerm*Background en XTerm*Foreground).
- Één van de prompts die in dit document wordt genoemd, maakt gebruik van de uitvoer van "jobs". Zoals later besproken zal worden, werkt de uitvoer van "jobs"naar een pipe niet goed onder Bash 2.0.2.
- ANSI escape sequences voor het verplaatsen van de cursor zijn niet in alle X-terminals geïmplementeerd. Dat wordt in een eigen sectie besproken.
- Een aantal aardig uitziende pseudo-graphics kan worden aangemaakt door gebruik te maken van een VGA-font in plaats van met de standaard Linux-fonts. Helaas zien deze effecten er afschuwelijk uit als je geen gebruik maakt van een VGA-font, en er is binnen een term geen manier waarop je kunt detecteren van welk soort font het gebruik maakt.
- Bash 2.0+ is uit. Er is een aantal nieuwe faciliteiten in opgenomen en het gedrag van een aantal is gewijzigd. Zaken die onder 1.1.4.7 werken, werken niet noodzakelijk hetzelfde onder 2.0+, en andersom.

1.5 Stuur me opmerkingen en aanbevelingen

Dit is een leerervaring voor mij. Ik ben tamelijk wat aan de weet gekomen over wat er kan worden gedaan om interessante en nuttige Bash-prompts te maken, maar ik heb je inbreng nodig om dit document te corrigeren en te verbeteren. Ik heb geprobeerd de door mij gedane aanbevelingen te controleren onder verschillende versies van Bash (2.0x en 1.14.7), maar laat me het weten als je enige incompatibiliteiten vindt.

De laatste versie van dit document zou altijd beschikbaar moeten zijn op *http://www.interlog.com/˜giles/bashprompt/*. Controleer dit alsjeblieft en mail me gerust aanbevelingen via *giles@interlog.com*.

Ik gebruik de Linux Documentatie Project HOWTO's bijna uitsluitend in het HTML-formaat, dus als ik dit vanuit SGML converteer (het bronformaat), is HTML het enige wat ik door en door controleer. Als er met andere formaten problemen zijn, kan ik dit niet weten en zou ik het waarderen als je er melding van zou maken.

1.6 Krediet

Tijdens het produceren van dit document heb ik veel afgekeken van het werk van het Bashprompt project op *http://bash.current.nu/*. Andere bronnen die ik heb gebruikt bestaan uit de *xterm Title mini-HOWTO* door Ric Lister, beschikbaar op *http://sunsite.unc.edu/LDP/HOWTO/mini/Xterm-Title.html*, *Ansi Prompts* door Keebler, beschikbaar op *http://www.ncal.verio.com/˜keebler/ansi.html*, *How to make a Bash Prompt Theme* door Stephen Webb, beschikbaar op *http://bash.current.nu/bash/HOWTO.html*, en *X ANSI Fonts* door Stumpy, beschikbaar op *http://home.earthlink.net/˜us5zahns/enl/ansifont.html*.

Ook van enorme hulp waren verscheidene conversaties en e-mails met Dan, een ex-medewerker van de Georgia College & State University, wiens kennis van Unix ver boven die van mij uitstijgt. Hij heeft me verscheidene uitstekende suggesties gegeven, en zijn ideeën hebben tot een aantal interessante prompts geleid.

Drie boeken die van zeer veel nut waren tijdens het programmeren van prompts zijn *Linux in a Nutshell* door Jessica Heckman Perry (O'Reilly, 1997), *Learning the Bash Shell* door Cameron Newham en Bill Rosenblatt (O'Reilly, 2nd. ed., 1998) en *Unix Shell Programming* door Lowell Jay Arthur (Wiley, 1986. Dit is de eerste editie, de vierde kwam uit in 1997).

1.7 Copyright en Disclaimer

2 Bash en Bash-Prompts

2.1 Wat is Bash?

Bash is een GNU-product, de "**Bourne Again SHellën afgeleid van de Bourne Shell. Op de meeste computers is het de standaard commandoregel-interface. Het munt uit in interactiviteit, ondersteunt het bewerken van de commandoregel, aanvullen en herroepen van commando's. Het biedt ook ondersteuning voor het configureren van prompts. De meeste mensen realiseren zich dit wel, maar weten niet hoeveel er mee kan worden gedaan.**

2.2 Wat Kan het Fijnafstemmen van je Bash-Prompt Voor je Doen?

De meeste Linux-systemen hebben een standaardprompt in één kleur (gewoonlijk grijs) welke je gebruikersnaam, de computernaam waarop je aan het werken bent en een indicatie van je huidige werkdirectory aangeeft. Dit is allemaal nuttige informatie, maar je kunt veel meer met de prompt. Alle soorten informatie kan worden getoond (tty-nummer, tijd, datum, load, aantal gebruikers, uptime ...) en de prompt kan gebruik maken van ANSI-kleuren, óf om het interessant te doen lijken, óf om bepaalde informatie er uit te laten springen. Ook de titelbalk van een Xterm kun je manipuleren om wat van deze informatie weer te geven.

2.3 Waarom zou je?

Buiten dat het er gaaf uit ziet, is het vaak handig om systeeminformatie bij te houden. Een idee, waarvan ik weet dat het sommige mensen aanspreekt, is dat het mogelijk is om de prompt op verschillende computers in andere kleuren te laten verschijnen. Als je op verscheidene verschillende computers diverse Xterms open hebt, of je bent geneigd te vergeten op welke computer je aan het werk bent en de verkeerde bestanden verwijdert (of de server afsluit in plaats van het werkstation), zul je dit een geweldige manier vinden om je er aan te herinneren op welke computer je werkt.

Zelf vind ik het handig om informatie over mijn computer en werkomgeving continu beschikbaar te hebben. En ik hou van de uitdaging uit te zoeken hoe de maximale hoeveelheid aan gegevens in de kleinst mogelijke ruimte te plaatsen waarbij de leesbaarheid behouden blijft.

2.4 De Eerste Stap

De weergave van de prompt wordt door de shell-variabele PS1 geregeld. Continuering van commando's wordt weergegeven door de PS2-string, welke op exact dezelfde wijze kan worden gewijzigd zoals hier besproken. Aangezien het beheren ervan exact hetzelfde is, en het niet zo ïnteressantïs, zal ik voornamelijk de PS1-string wijzigen. (Er zijn ook PS3- en PS4-strings. Deze worden door de gemiddelde gebruiker nooit opgemerkt. Zie de man page van Bash als je in hun doeleinden bent geïnteresseerd). Om de wijze hoe de prompt er uitziet te wijzigen, verander je de PS1-variabele. Je kunt de PS1-strings direct achter de prompt invoeren om ermee te experimenteren en de resultaten onmiddellijk te zien (dit heeft alleen effect op je huidige sessie en de wijzigingen worden ongedaan gemaakt als je uitlogt). Als je een blijvende wijziging van de prompt wilt, kijk dan in de sectie onder 2.6 (Permanent instellen van de PS?-Strings).

Voordat we gaan beginnen is het belangrijk er aan te denken, dat de PS1-string net als elke andere omgevingsvariabele in de omgeving is opgeslagen. Als je het achter de commandoregel wijzigt, zal je prompt veranderen. Voordat je enige veranderingen aanbrengt, kun je de huidige prompt in een andere omgevingsvariabele bewaren:

```
[giles@nikola giles]$ SAVE=$PS1
[giles@nikola giles]$
```

De eenvoudigste prompt zou uit een enkel teken kunnen bestaan, zoals:

```
[giles@nikola giles]$ PS1=$
$ls
bin   mail
$
```

Door ze op de commandoregel in te voeren, wordt de beste manier getoond waarop je met basisprompts kunt experimenteren. Merk op dat de tekst die door de gebruiker werd ingetikt, onmiddellijk achter de prompt verschijnt. Ik geef de voorkeur aan het gebruik van:

```
$PS1="$ "
$ ls
bin   mail
$
```

waardoor een spatie achter de prompt wordt geforceerd, wat het beter leesbaar maakt. Roep gewoon de variabele op die je hebt opgeslagen om de oorspronkelijke prompt te herstellen:

```
$ PS1=$SAVE
[giles@nikola giles]$
```

2.5 Bash Prompt Escape Sequences

Er worden door de Bash-shell een heleboel escape sequences geboden om in de prompt in te voegen. Vanuit de man page van Bash 2.02:

```
Wanneer interactief uitgevoerd, toont bash de primaire prompt PS1 als
het gereed is een commando in te lezen, en de secundaire prompt PS2 als
het meer invoer nodig heeft om het commando te completeren.
Met Bash is het mogelijk deze prompt-strings te wijzigen door een aantal
speciale tekens beginnend met een escape-teken, de backslash, in te voegen,
die als volgt zijn gecodeerd:
       \a     een ASCII bell teken (07)
       \d     de datum  in  "Dag van de Week Maand Datum" formaat
              (d.w.z., "Tue May 26")
       \e     een ASCII escape character (033)
       \h     de hostname tot aan de eerste `.'
       \H     de hostname
       \n     nieuwe regel
       \r     carriage return
       \s     de naam van de shell, de  basename  van  $0
              (het deel volgend na de laatste slash)
       \t     de huidige tijd in het 24-uurs HH:MM:SS formaat
       \T     de huidige tijd in het 12-uurs HH:MM:SS formaat
       \@     de huidige tijd in 12-uurs am/pm formaat
       \u     de gebruikersnaam van de huidige gebruiker
       \v     de versie van bash (d.w.z. 2.00)
       \V     de  release  van  bash, versie + patchlevel
              (d.w.z. 2.00.0)
       \w     de huidige werkdirectory
       \W     de basename van de huidige werkdirectory
       \!     het historienummer van dit commando
       \#     het commandonummer van dit commando
       \$     als de effectieve UID is 0, een #, anders een $
       \nnn   het teken corresponderend met het octale nummer nnn
       \\     een backslash
       \[     begin een reeks niet-afdrukbare tekens,
              die kunnen worden gebruikt om een terminal besturingsreeks
              in de prompt in te sluiten
       \]     eindig een reeks niet-afdrukbare tekens
```

Verder gaan bij waar we waren gebleven:

```
[giles@nikola giles]$ PS1="\u@\h \W> "
giles@nikola giles> ls
bin   mail
giles@nikola giles>
```

Dit lijkt voor de meeste Linux-distributies op de standaard. Ik wilde dat het er iets anders uitzag, dus ik wijzigde dit in:

```
giles@nikola giles> PS1="[\t][\u@\h:\w]\$ "
[21:52:01][giles@nikola:~]$ ls
bin   mail
[21:52:15][giles@nikola:~]$
```

2.6 Permanent Instellen van de PS?-Strings

Mensen en distributies plaatsen hun PS?-strings op verschillende locaties. De meest gebruikelijke locaties zijn /etc/profile, /etc/bashrc, ~/.bash_profile, en ~/.bashrc . Johan Kullstam (*johan19@idt.net*) schrijft:

de PS1-string zou in .bashrc moeten worden ingesteld. Dit omdat niet-interactieve bash'es buiten hun boekje gaan door PS1 te unsetten. De man page van bash vertelt hoe de aanwezigheid of afwezigheid van PS1 een goede manier is om er achter te komen of men zich in een interactieve dan wel niet-interactieve (b.v. script) bash-sessie bevindt.

de manier waarop ik dit realiseerde is, dat startx een bash-script is. Dit betekent dat startx je prompt zal wissen. als je PS1 in .profile (of .bash_profile) instelt, op de console inlogt en X via startx opstart, wordt je PS1 in het proces gewist en blijf je achter met de standaardprompt.

één oplossing is xterms en rxvts met de -ls optie op te starten, om ze te dwingen .profile in te lezen. maar elke keer dat er een shell wordt aangeroepen via een niet-interactief shell-script gaat de tussenliggende PS1 verloren. system(3) gebruikt sh -c welke, indien sh bash is, PS1 zal killen. een betere manier is om de PS1 definitie in .bashrc te plaatsen. deze wordt iedere keer gelezen als bash start en is de plaats waarin interactieve zaken (bv PS1) zouden moeten staan.

daarom zou moeten worden benadrukt dat PS1=..blah.. in .bashrc behoort te staan en niet in .profile.

Ik probeerde het probleem dat hij uitlegde te herhalen en ik constateerde een ander probleem: mijn PROMPT_COMMAND variabele (welke later zal worden geïntroduceerd) was verdwenen. Mijn kennis op dit gebied is wat zwak, dus ik ga ervoor wat Johan zegt.

3 Bash Programmeren en Shell-Scripts

3.1 Variabelen

Ik ga hier niet alle details betreffende Bash-scripts in een sectie van deze HOWTO proberen uit te leggen, maar alleen de details die betrekking hebben op prompts. Als je meer wilt weten over shell-programmeren en van Bash in het algemeen, beveel ik je van harte het boek *Learning the Bash Shell* aan, van Cameron Newham en Bill Rosenblatt (O'Reilly, 1998). Vreemd genoeg is mijn kopie van het boek nogal versleten. Nogmaals, ik ga ervan uit dat je al tamelijk wat weet over Bash. Je kunt deze sectie overslaan als je alleen op zoek bent naar de grondbeginselen, maar onthoud het en kom terug als je wat verder bent gekomen.

Variabelen worden in Bash bijna net als in iedere andere programmeertaal toegekend:

```
testvar=5
foo=zen
bar="bash prompt"
```

Aanhalingstekens zijn alleen nodig in een toekenning als een spatie (of speciaal teken, hetgeen beknopt wordt besproken) onderdeel uitmaakt van de variabele.

Er wordt iets anders naar variabelen gerefereerd dan hoe ze worden toegekend:

```
> echo $testvar
5
> echo $foo
zen
> echo ${bar}
bash prompt
> echo $NietToegekend

>
```

Er kan naar een variabele worden gerefereerd als $bar of ${bar}. De accolades zijn handig als het onduidelijk is waarnaar zal worden gerefereerd. Als ik $barley schrijf, bedoel ik dan in werkelijkheid ${bar}ley of ${barley}? Merk ook op dat wanneer er naar een waarde wordt gerefereerd die niet is toegekend, er geen fout wordt gegenereerd. In plaats daarvan wordt er niets geretourneerd.

3.2 Aanhalingstekens en Speciale Tekens

Als je speciale tekens in een variabele wilt opnemen, zul je het anders aan moeten halen (quoten):

```
> newvar=$testvar
> echo $newvar
5
> newvar="$testvar"
> echo $newvar
5
> newvar='$testvar'
> echo $newvar
$testvar
> newvar=\$testvar
> echo $newvar
$testvar
>
```

Het dollar-teken is niet het enige speciale teken voor de Bash-shell, maar het is een eenvoudig voorbeeld. Een interessante stap die we kunnen nemen om gebruik te maken van de toekenning van een variabelenaam aan een andere variabelenaam is door eval te gebruiken om naar de opgeslagen variabelenaam te verwijzen:

```
> echo $testvar
5
> echo $newvar
$testvar
> eval echo $newvar
5
>
```

Normaal gesproken maakt de shell slechts één substituties-slag in de uitdrukking welke hij evalueert. Als je zegt echo $newvar, zal de shell slechts zover gaan dat het vaststelt dat $newvar gelijk is aan de tekststring $testvar. Hij zal niet evalueren waaraan $testvar gelijk is. eval forceert die evaluatie.

3.3 Commando Substitutie

Ik gebruik in bijna alle situaties in dit document de $(<commando>) conventie voor commando-substitutie: dat wil zeggen,

```
$(date +%H%M)
```

betekent "vervang hier de uitvoer van het date +%H%M commando."Dit werkt in Bash 2.0+. In een aantal oudere versies van Bash, van voor 1.14.7, kan het zijn dat je enkele aanhalingstekens openen, (`date +%H%M`) moet gebruiken. Enkele aanhalingstekens openen kunnen in Bash 2.0+ worden gebruikt, maar zullen geleidelijk verdwijnen ten gunste van $(), welke beter is te nesten. Als je een eerdere versie van Bash gebruikt, kun je de aanhalingstekens openen meestal vervangen op die plaatsen waar je $() ziet. Als de commando-substitutie door escape-tekens is omsloten, (d.w.z. \$(commando)), gebruik dan backslashes om BEIDE aanhalingstekens openen te escapen (d.w.z. \'commando\').

3.4 Niet Afdrukbare Tekens in Prompts

Veel van de wijzigingen die kunnen worden aangebracht in Bash-prompts welke in deze HOWTO worden besproken, maken gebruik van niet-afdrukbare tekens. Het wijzigen van de kleur van de prompttekst, het wijzingen van de titelbalk van een Xterm en het verplaatsen van de cursorpositie vereisen alle niet-afdrukbare tekens.

Als je een zeer eenvoudige prompt bestaande uit een groter-dan teken en een spatie wilt:

```
[giles@nikola giles]$ PS1='> '
>
```

Dit is slechts een prompt bestaande uit twee tekens. Als ik het zodanig wijzig dat het groter-dan teken heldergeel is (kleuren worden in een eigen sectie beschreven):

```
> PS1='\033[1;33m>\033[0m '
>
```

Dit werkt prima, totdat je een lange commandoregel intikt. Ook al bestaat de prompt uit nog maar twee afdrukbare tekens (een groter-dan-teken en een spatie), de shell denkt dat deze prompt uit elf tekens bestaat (Ik denk dat het '\033', '[1' en '[0' ieder als één teken telt). Je kunt dit zien als je een echt lange commandoregel intikt. Je zult bemerken dat de shell de tekst op de volgende regel plaatst, voordat het de rand van de terminal bereikt. In de meeste gevallen gaat dit niet goed. Dit komt doordat het voor de shell onduidelijk is hoelang de feitelijke lengte van de prompt is.

Dus gebruik in plaats daarvan:

```
> PS1='\[\033[1;33m\]>\[\033[0m\] '
```

Dit is wat complexer, maar het werkt. Commandoregels worden juist afgebroken. De '\033[1;33m' waarmee de kleur geel wordt begonnen, wordt omsloten door teksthaken. Dit, inclusief de teksthaken zelf, is een niet-afdrukbaar teken. Hetzelfde wordt gedaan met de '\033[0m' waarmee het einde van de kleur geel wordt aangegeven.

3.5 Commando's in een Bestand

Als een bestand met commando's wordt aangeroepen (door het op de commandoregel typen van source filename of . filename), worden de regels code in het bestand uitgevoerd alsof ze op de commandoregel werden ingetypt. Dit is vooral nuttig bij complexe prompts, omdat het mogelijk is ze in bestanden op te slaan en de commando's in het bestand aan te roepen door het aanroepen van het bestand.

In voorbeelden zul je vaak aantreffen dat ik aan het begin van bestanden met functies de regel #!/bin/bash heb toegevoegd. Dit is niet noodzakelijk als je een bestand met commando's aanroept, net als dat het niet noodzakelijk is als je een chmod +x toepast op een bestand waaruit de commando's worden gelezen. Ik doe dit om ervoor te zorgen dat Vim (editor van mijn keuze, geen gescheld alsjeblieft, jij gebruikt wat jij wilt) denkt dat ik een shell-script aan het wijzigen ben, en daardoor kleuren syntax highlighting aanzet.

3.6 Functies, Aliassen, en de Omgeving

Zoals al eerder gezegd, worden de PS1, PS2, PS3, PS4, en PROMPT_COMMAND alle in de Bash-omgeving opgeslagen. Voor degenen onder ons met een DOS-achtergrond, is het een afschuwelijk idee om grote hoeveelheden code in de omgeving te plaatsen, omdat die DOS-omgeving klein was en niet precies op de juiste manier toenam. Er zijn waarschijnlijk praktische grenzen aan wat je in de omgeving kan en zou moeten plaatsen, maar ik ken ze niet. We hebben het hier waarschijnlijk over enkele grootte-orden meer dan wat DOS-gebruikers gewend zijn.

Zoals Dan het deed: "In mijn interactieve shell heb ik 62 aliassen en 25 functies. Mijn stelregel is dat als ik iets alleen voor interactief gebruik nodig heb en het eenvoudig in bash te schrijven is, ik er een shell-functie van maak (in de veronderstelling dat het eenvoudig als een alias kan worden uitgedrukt). Als mensen zich druk maken over het geheugen dan is dat bij gebruik van bash niet nodig. Bash is één van de grootste programma's die ik op mijn linux box draai (buiten Oracle). Draai top zo nu en dan en druk op 'M' om op geheugen te sorteren en zie hoe dicht bash bovenaan de lijst staat. Jandorie, het is groter dan sendmail! Zeg hun om ash of iets dergelijks aan te schaffen."

Ik gok erop dat hij alleen van de console gebruikmaakte toen hij het draaien van X en X-apps probeerde. Ik heb heel wat liggen dat groter is dan Bash. Maar het idee is hetzelfde: de omgeving is iets om te gebruiken, en maak je geen zorgen als het te veel wordt.

Ik riskeer afkeuring van Unix-goeroes wanneer ik dit zeg (voor de misdaad het al te eenvoudig te maken), maar functies zijn in principe kleine shell-scripts die om efficiëncy-redenen in de omgeving worden geladen. Dan weer aanhalend: "Shell functies zijn ongeveer zo efficiënt als ze kunnen zijn. Het is bij benadering equivalent aan het inlezen van een bewaarde bash/bourne Shell-script zonder dat bestands-I/O nodig is, aangezien de functie zich al in het geheugen bevindt. De shell-functies worden typisch geladen vanuit [.bashrc of .bash_profile] afhankelijk van of je ze alleen in de initiële shell wilt of ook in de subshells. Dit in tegenstelling tot het uitvoeren van een shell-script. Je shell splitst zich af en de child doet een exec. Eventueel wordt het pad doorzocht. De kernel opent het bestand en onderzoekt voldoende bytes om vast te stellen hoe het bestand uitgevoerd moet worden. In het geval van een shell-script moet er een shell worden gestart met de naam van het script als argument. De shell opent vervolgens het bestand, lees het in en voert de opdrachten uit. Vergeleken met een shell-functie, kan al het andere dan het uitvoeren van de opdrachten worden aangemerkt als onnodige overhead."

Aliassen zijn simpel aan te maken:

```
alias d="ls --color=tty --classify"
alias v="d --format=long"
alias rm="rm -i"
```

Alle argumenten die je aan de alias doorgeeft worden aan de commandoregel van het ge-alias-te commando doorgegeven (ls in de eerste twee gevallen). Merk op dat aliassen kunnen worden genest en ze kunnen worden gebruikt om een gewoon unix-commando zich op een andere manier te laten gedragen. (Ik ben het met het argument eens dat je de laatste soort aliassen niet zou moeten gebruiken.Als je in de gewoonte vervalt er op te vertrouwen dat "rm *"je vraagt of je wel zeker bent, kun je belangrijke bestanden op een systeem kwijtraken die geen gebruik maakt van je alias).

Functies worden gebruikt voor complexere programmastructuren. Als algemene regel moet je een alias gebruiken voor alles dat in één regel kan worden gedaan. Functies verschillen van shell-scripts in die zin dat ze in de omgeving worden geladen zodat ze sneller werken. Nogmaals, als algemene regel zou je je functies relatief klein moeten houden, en ieder shell-script dat relatief groot wordt zou een shell-script moeten blijven in plaats dat je het omzet in een functie. Je beslissing om iets als een functie te laden zal ook afhangen van hoe vaak je het gebruikt. Als je een klein shell-script niet vaak gebruikt, laat het dan als een shell-script. Als je het vaak gebruikt, zet het dan om in een functie.

Om het gedrag van ls te wijzigen, zou je iets kunnen doen als het volgende:

```
function lf
{
    ls --color=tty --classify $*
    echo "$(ls -l $* | wc -l) files"
}
```

Dit zou makkelijk als een alias kunnen worden ingesteld, maar ter wille van het voorbeeld zullen we er een functie van maken. Als je de tekst typt in een tekstbestand en je past een source toe op dat bestand, zal de functie in je omgeving staan en onmiddellijk beschikbaar zijn op de commandoregel zonder de overhead van een shell-script zoals voorheen werd aangegeven. Het nut hiervan wordt duidelijker als je overweegt meer functionaliteit aan de functie van hierboven toe te voegen, zoals het gebruik van een if-opdracht om speciale code uit te voeren als er links in de listing worden gevonden.

4 Externe Commando's

4.1 PROMPT_COMMAND

Bash voorziet in een andere omgevingsvariabele genaamd PROMPT_COMMAND. De inhoud van deze variabele wordt uitgevoerd als een regulier Bash-commando net voordat Bash een prompt toont.

```
[21:55:01][giles@nikola:~] PS1="[\u@\h:\w]\$ "
[giles@nikola:~] PROMPT_COMMAND="date +%H%M"
2155
[giles@nikola:~] d
bin   mail
2156
[giles@nikola:~]
```

Wat hierboven gebeurde is dat ik PS1 zo wijzigde dat de escape sequence `\t` er niet langer in was opgenomen, dus de tijd maakte niet langer onderdeel uit van de prompt. Toen gebruikte ik `date +%H%M` om de tijd in een beter formaat te tonen. Maar het verschijnt op een andere regel dan de prompt. Dit in orde brengend met `echo -n ...` zoals hieronder getoond werkt met Bash 2.0+, maar schijnt niet met Bash 1.14.7 te werken. Blijkbaar wordt de prompt op een andere manier getekend. De volgende methode resulteert in een overlappende tekst.

```
2156
[giles@nikola:~] PROMPT_COMMAND="echo -n [$(date +%H%M)]"
[2156][giles@nikola:~]$
[2156][giles@nikola:~]$ d
bin   mail
[2157][giles@nikola:~]$ unset PROMPT_COMMAND
[giles@nikola:~]
```

`echo -n ...` beheert de uitvoer van het `date` commando en onderdrukt de opvolgende newline, waardoor de prompt geheel op één regel kan verschijnen. Aan het einde maakte ik gebruik van het `unset` commando om de omgevingsvariabele PROMPT_COMMAND te verwijderen.

4.2 Externe Commando's in de Prompt

Je kunt tevens direct de uitvoer van reguliere Linux-commando's in de prompt gebruiken. Uiteraard wil je niet al te veel materiaal invoegen, anders zal het een grote prompt creëren. Je zult een snel commando willen omdat het iedere keer dat je prompt op het scherm verschijnt zal worden uitgevoerd.Onderbrekingen in de weergave van je prompt terwijl je aan het werken bent kan zeer ergerlijk zijn. (In tegenstelling tot het vorige voorbeeld waar dit veel op lijkt, werkt dit wel met Bash 1.14.7).

```
[21:58:33][giles@nikola:~]$ PS1="[\$(date +%H%M)][\u@\h:\w]\$ "
[2159][giles@nikola:~]$ ls
bin   mail
[2200][giles@nikola:~]$
```

Let goed op de backslash voor het dollar-teken van de commando substitutie. Zonder die backslash wordt het externe commando exact éénmaal uitgevoerd, als de PS1-string in de omgeving wordt gelezen. Voor deze prompt zou dat betekenen dat het ongeacht hoelang de prompt in gebruik was, dezelfde tijd zou tonen. De backslash schermt de inhoud van $() af van onmiddellijke shell-interpretatie , dus "date"wordt iedere keer dat een prompt wordt gegenereerd, aangeroepen.

Linux wordt met veel kleine utility's geleverd, zoals date, grep, of wc waarmee je data kunt manipuleren. Als je merkt dat je probeert complexe combinaties van deze programma's binnenin een prompt aan te maken, kan het eenvoudiger zijn om er een eigen alias, functie of shell-script van te maken en het vanuit de prompt aan te roepen. Escape sequences zijn vaak

vereist in bash shell-scripts om er zeker van te zijn dat shell-variabelen op de juiste tijd worden uitgewerkt (zoals te zien in het date-commando hierboven): dit komt op een ander niveau binnen de prompt PS1-regel naar boven, en het voorkomen ervan door het aanmaken van functies is een goed idee.

Een voorbeeld van een klein shell-script dat binnen een prompt wordt gebruikt, wordt hieronder gegeven:

```
#!/bin/bash
#     lsbytesum - geef het totaal aantal bytes in een directorylisting
TotalBytes=0
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \n$TotalBytes/1048576 \nquit" | bc)
echo -n "$TotalMeg"
```

Ik heb dit tijden zowel als een functie als in een shell-script in mijn ~/bin directory bewaard, die zich in mijn path bevindt. Gebruikt in een prompt:

```
[2158][giles@nikola:~]$ PS1="[\u@\h:\w (\$(lsbytesum) Mb)]\$ "
[giles@nikola:~ (0 Mb)]$ cd /bin
[giles@nikola:/bin (4.498 Mb)]$
```

4.3 Wat in je Prompt te Plaatsen

Je zult zien dat ik mijn gebruikersnaam, computernaam, de tijd en huidige directorynaam in mijn meeste prompts gebruik. Met uitzondering van de tijd, zijn dit zeer standaard-items die in een prompt te vinden zijn, en tijd is waarschijnlijk de eerstvolgende algemene aanvulling. Maar wat je opneemt is geheel een kwestie van smaak. Hier zijn wat voorbeelden van mensen die ik ken om je op ideeën te brengen.

Dan's prompt is minimaal, maar zeer effectief, vooral voor de manier waarop hij werkt.

```
[giles@nikola:~]$ cur_tty=$(tty | sed -e "s/.*tty\(.*\)/\1/")
[giles@nikola:~]$ echo $cur_tty
p4
[giles@nikola:~]$ PS1="\!,$cur_tty,\$?\$ "
1095,p4,0$
```

Dan houdt er niet van als de huidige werkdirectory de prompt drastisch van grootte doet veranderen, als je je door de directorystructuur verplaatst, dus onthoudt hij dit (of typt "pwd"). Hij leerde Unix met csh en tcsh, dus maakt hij veelvuldig gebruik van de commando-historie (iets dat velen van ons betreurden dat Bash dit niet deed). Dus het eerste item in de prompt is het historie-nummer. Het tweede item bestaat uit de veelbetekenende tekens van de tty (de uitvoer van "ttyïs er met sed uitgeknipt), een item dat handig kan zijn voor "screen"gebruikers. Het derde item is de exit waarde van het laatste commando/pipeline (merk op dat deze weergave nutteloos is door enig commando uitgevoerd binnen de prompt. Je zou dit echter op kunnen lossen door het in een variabele af te vangen en het terug te spelen). Als laatste is het "\$ëen dollar-teken voor een reguliere gebruiker en verandert in een hekje ("#") als de gebruiker root is.

Torben Fjerdingstad (tfj@fjerdingstad.dk) schreef me om me te vertellen dat hij zijn jobs vaak uitstelt en ze dan vaak vergeet. Hij gebruikt zijn prompt om zichzelf aan uitgestelde taken (suspended jobs) te herinneren:

```
[giles@nikola:~]$ function jobcount {
> jobs|wc -l| awk '{print $1}'
> }
[giles@nikola:~]$ export PS1='\W[`jobcount`]# '
giles[0]# man ls &
[1] 4150

[1]+  Stopped (tty output)    man ls
giles[1]#
```

Torben gebruikt awk om de blanco's uit de uitvoer van wc te halen, terwijl ik gebruik zou hebben gemaakt van sed of tr, niet omdat die beter zijn, maar omdat ik daar meer bekend mee ben. Er zijn waarschijnlijk ook nog andere manieren. Torben plaatst ook nog enkele aanhalingstekens om zijn PS1-string, wat voorkomt dat Bash de aanhalingstekens openen (') onmiddellijk interpreteert. Dus hoeft hij ze niet, zoals ik noemde, te escapen.

NOOT: Er is een bekende bug in Bash 2.02 die er voor zorgt dat een jobs commando (een intern shell-commando) naar een pipe niets retourneert. Als je het bovenstaande onder Bash 2.02 probeert, zul je altijd een "0"terug krijgen, ongeacht hoeveel jobs je hebt uitgesteld. Dit probleem is in 2.03 hersteld.

5 Xterm Titelbalk Manipulaties

Niet-afdrukbare escape sequences kunnen worden gebruikt om interessante effecten in prompts te produceren. Om deze escape sequences te gebruiken moet je ze omsluiten door \[en \] (zoals besproken in 3.4 (Niet Afdrukbare Tekens in Prompts)), waarbij je Bash vertelt dit materiaal te negeren als het de grootte van de prompt aan het berekenen is. Als je deze scheidingstekens niet insluit, resulteert dat in code waarbij de cursor onjuist wordt geplaatst, omdat het de werkelijke grootte van de prompt niet kent. Escape sequences moeten ook worden voorafgegaan door \033[in Bash van voor versie 2, of door \033[of \e[in latere versies.

Als je probeert de titelbalk van je Xterm met je prompt te wijzigen onder de console, zul je rommel in je prompt produceren. Test de TERM omgevingsvariabele om aan te laten geven of je prompt in een Xterm is, om dit te voorkomen.

```
function proml
{
case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=''
        ;;
esac
```

```
PS1="${TITLEBAR}\
[\$(date +%H%M)]\
[\u@\h:\w]\
\$ "
PS2='> '
PS4='+ '
}
```

Dit is een functie die in ~/.bashrc kan worden opgenomen. De functienaam zou dan moeten worden aangeroepen om de functie uit te voeren. De functie wordt net als de PS1-string in de omgeving opgeslagen. Zodra de PS1-string door de functie is ingesteld, kun je de functie uit de omgeving verwijderen met `unset proml`. Aangezien de prompt niet verplaatst kan worden vanuit de console naar een Xterm, wordt de TERM-variabele niet iedere keer getest als de prompt wordt gegenereerd. Ik gebruikte vervolgmarkeringen (backslashes) in de definitie van de prompt, waardoor het mogelijk wordt het op meerdere regels te continueren. Dit verbetert de leesbaarheid, waardoor het eenvoudiger is wijzigingen aan te maken en fouten op te sporen.

Ik definieer dit als een functie omdat het Bashprompt package zo met prompts omgaat (8 (Het Bash Prompt Package) wordt later in dit document besproken). Het is niet de enige manier om 't te doen, maar het werkt goed. Naarmate de door jouw gebruikte prompts steeds complexer worden, wordt het steeds lastiger ze achter de prompt in te typen en zal het praktischer zijn ze in één of ander tekstbestand aan te maken. Bewaar in dit geval het hierbovengenoemde als een tekstbestand genaamd "promlöm dit achter de prompt te testen.

Je kunt er als volgt mee werken:

```
[giles@nikola:/bin (4.498 Mb)]$ cd          -> Ga naar waar je de prompt op
                                               wilt slaan
[giles@nikola:~ (0 Mb)]$ vi proml           -> Wijzig het promptbestand
...                                         -> Voer de gegeven tekst van
                                               hierboven in
[giles@nikola:~ (0 Mb)]$ source proml       -> Lees de promptfunctie in
[giles@nikola:~ (0 Mb)]$ proml              -> Voer de promptfunctie uit
```

De eerste stap in het aanmaken van deze prompt is het testen of de te starten shell een xterm is of niet. Als dit niet zo is, wordt de shell-variabele (${TITLEBAR}) gedefinieerd. Het bestaat uit de juiste escape sequences en \u@\h:\w, waarmee de <user>@<machine>:<werkdirectory> in de Xterm titelbalk worden geplaatst. Dit is vooral nuttig met geminimaliseerde Xterms, omdat ze daardoor sneller identificeerbaar zijn. Het andere materiaal in deze prompt zou bekend moeten zijn van de vorige prompts die we hebben aangemaakt.

De enige keerzijde aan het manipuleren van de titelbalk van een Xterm zoals deze, komt voor wanneer je op een systeem inlogt waarop je de titelbalk hack niet hebt ingesteld. De Xterm zal de informatie continu laten zien van het vorige systeem dat de titelbalk hack had.

6 ANSI Escape Sequences: Kleuren en Cursorverplaatsing

6.1 Kleuren

Zoals hierboven aangegeven moeten niet-afdrukbare tekens worden omsloten door \[\033[en \]. Voor kleuren escape sequences, zouden ze ook gevolgd moeten worden door een kleine letter m.

Als je de volgende prompts in een xterm uitprobeert en bemerkt dat je geen kleuren benoemd ziet, controleer dan het bestand ~/.Xdefaults (en mogelijk zijn broeders) op regels als "XTerm*Foreground: BlanchedAlmond". Maak er een commentaarregel van door er een uitroepteken ("!") voor te plaatsen. Natuurlijk zal dit ook afhangen van welke terminal-emulator je gebruik maakt. Dit is de meest waarschijnlijke plaats waar de voorgrondkleuren van je term zouden kunnen zijn overschreven.

Om blauwe tekst in de prompt op te nemen:

```
PS1="\[\033[34m\][\$(date +%H%M)][\u@\h:\w]$ "
```

Het probleem met deze prompt is dat de blauwe kleur die met kleurcode 34 begint nooit naar de reguliere kleur wordt teruggezet, dus alle tekst die je achter de prompt intikt staat nog steeds in de kleur van de prompt. Dit is ook een donkere vorm blauw dus het zou kunnen helpen het te combineren met de code bold:

```
PS1="\[\033[1;34m\][\$(date +%H%M)][\u@\h:\w]$\[\033[0m\] "
```

De prompt staat nu in lichtblauw en het eindigt door het terugschakelen van de kleur naar niets (welke voorgrondkleur je voorheen ook had).

Dit is de rest van de kleur-equivalenten:

```
Zwart       0;30     Donkergrijs  1;30
Blauw       0;34     Lichtblauw   1;34
Groen       0;32     Lichtgroen   1;32
Cyaan       0;36     Lichtcyaan   1;36
Rood        0;31     Lichtrood    1;31
Violet      0;35     Lichtviolet  1;35
Bruin       0;33     Geel         1;33
Lichtgrijs  0;37     Wit          1;37
```

Daniel Dui (ddui@iee.org) wijst erop dat om strikt accuraat te zijn. We moeten aangeven dat de lijst hierboven voor kleuren onder de console is. In een xterm, is de code 1;31 niet "Lichtrood,"maar "Benadrukt Rood."Dit geldt voor alle kleuren.

Je kunt ook achtergrondkleuren verkrijgen door 44 te gebruiken voor de Blauwe achtergrond, 41 voor een Rode achtergrond, enz. Er zijn geen benadrukte achtergrondkleuren. Er kunnen combinaties worden gebruikt, zoals Lichtrode tekst op een Blauwe achtergrond: \[\033[44;1;31m\], alhoewel het apart instellen van de kleuren beter schijnt te werken (bv. \[\033[44m\]\[\033[1;31m\]). Andere beschikbare codes zijn 4: Underscore, 5: Blink, 7: Inverse, en 8: Concealed.

Ter zijde: Veel mensen (inclusief mezelf) hebben zeer veel bezwaar tegen het "blink"kenmerk. Gelukkig werkt het niet in alle terminal emulators zover ik weet, maar het werkt nog steeds op de console. En mocht je het je afvragen (zoals ik deed) "Waarvoor een 'Concealed' kenmerk te gebruiken?!- Ik zag dat het in een voorbeeld shell-script (geen prompt) werd gebruikt om iemand toe te staan een wachtwoord in te typen zonder dat het naar het scherm werd teruggekaatst.

Gebaseerd op een prompt genaamd ëlite2ïn het Bashprompt package (wat ik heb gewijzigd om het beter te laten werken op een standaardconsole, in plaats van met de speciale vereiste xterm-fonts om de originele goed te kunnen zien), is dit een prompt die ik veel heb gebruikt:

```
function elite
{

local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local CYAN="\[\033[0;36m\]"
local LIGHT_CYAN="\[\033[1;36m\]"

case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=""
        ;;
esac

local GRAD1=$(tty|cut -d/ -f3)
PS1="$TITLEBAR\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\u$GRAY@$CYAN\h\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\#$GRAY/$CYAN$GRAD1\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\$(date +%H%M)$GRAY/$CYAN\$(date +%d-%b-%y)\
$LIGHT_CYAN)$CYAN-$GRAY-\
$LIGHT_GRAY\n\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\$$GRAY:$CYAN\w\
$LIGHT_CYAN)$CYAN-$GRAY-$LIGHT_GRAY "
PS2="$LIGHT_CYAN-$CYAN-$GRAY-$LIGHT_GRAY "
}
```

Voor de leesbaarheid definieer ik de kleuren als tijdelijke shell-variabelen. Het is makkelijker om er mee te werken. De "GRAD1"variabele is een controle om vast te stellen op welke terminal je je bevindt. Zoals de test om vast te stellen of je in een Xterm aan het werken bent, hoeft het slechts éénmaal te worden gedaan. De prompt ziet er ongeveer zo uit, behalve dan in kleur:

```
--(giles@nikola)-(75/ttyp7)-(1908/12-Oct-98)--
--($:~/tmp)--
```

Om mezelf eraan te helpen herinneren welke kleuren beschikbaar zijn, schreef ik het volgende script die alle kleuren naar het scherm echoot:

```
#!/bin/bash
#
#   Dit bestand echoot veel kleurcodes naar de terminal om te demonstreren
#   wat beschikbaar is. Iedere regel bestaat uit één kleur met
```

```
#   als achtergrond zwart en grijs, met de code in het midden. Geverifieerd
#   dat het met een witte, zwarte en grijze achtergrond werkt (2 Dec 98).
#
echo "  Op Lichtgrijs::        Op Zwart:"
echo -e "\033[47m\033[1;37m  Wit        \033[0m\
 1;37m \
\033[40m\033[1;37m  Wit        \033[0m"
echo -e "\033[47m\033[37m  Lichtgrijs   \033[0m\
   37m \
\033[40m\033[37m  Lichtgrijs   \033[0m"
echo -e "\033[47m\033[1;30m  Grijs         \033[0m\
 1;30m \
\033[40m\033[1;30m  Grijs         \033[0m"
echo -e "\033[47m\033[30m  Zwart        \033[0m\
   30m \
\033[40m\033[30m  Zwart        \033[0m"
echo -e "\033[47m\033[31m  Rood          \033[0m\
   31m \
\033[40m\033[31m  Rood          \033[0m"
echo -e "\033[47m\033[1;31m  Lichtrood    \033[0m\
 1;31m \
\033[40m\033[1;31m  Lichtrood    \033[0m"
echo -e "\033[47m\033[32m  Groen        \033[0m\
   32m \
\033[40m\033[32m  Groen        \033[0m"
echo -e "\033[47m\033[1;32m  Lichtgroen  \033[0m\
 1;32m \
\033[40m\033[1;32m  Lichtgroen  \033[0m"
echo -e "\033[47m\033[33m  Bruin        \033[0m\
   33m \
\033[40m\033[33m  Bruin        \033[0m"
echo -e "\033[47m\033[1;33m  Geel       \033[0m\
 1;33m \
\033[40m\033[1;33m  Geel       \033[0m"
echo -e "\033[47m\033[34m  Blauw         \033[0m\
   34m \
\033[40m\033[34m  Blauw         \033[0m"
echo -e "\033[47m\033[1;34m  Lichtblauw   \033[0m\
 1;34m \
\033[40m\033[1;34m  Lichtblauw   \033[0m"
echo -e "\033[47m\033[35m  Violet       \033[0m\
   35m \
\033[40m\033[35m  Violet       \033[0m"
echo -e "\033[47m\033[1;35m  Rose         \033[0m\
 1;35m \
\033[40m\033[1;35m  Rose         \033[0m"
echo -e "\033[47m\033[36m  Cyaan         \033[0m\
   36m \
\033[40m\033[36m  Cyaan         \033[0m"
echo -e "\033[47m\033[1;36m  Lichtcyaan   \033[0m\
```

```
 1;36m \
\033[40m\033[1;36m  Lichtcyaan   \033[0m"
```

6.2 Cursorverplaatsing

Met ANSI escape sequences kun je de cursor naar wens op het scherm verplaatsen. Dit is nuttiger voor gebruikersinterfaces met volledig scherm, gegenereerd door shell-scripts, maar kan ook in prompts worden gebruikt. De verplaatsings escape sequences zijn als volgt:

```
    - Positioneer de cursor:
      \033[<L>;<C>H
         Of
      \033[<L>;<C>f
      plaats de cursor op regel L en kolom C.
    - Verplaats de cursor N regel(s) omhoog:
      \033[<N>A
    - Verplaats de cursor N regel(s) naar beneden:
      \033[<N>B
    - Verplaats de cursor N kolom(men) vooruit:
      \033[<N>C
    - Verplaats de cursor N kolom(men) terug:
      \033[<N>D

    - Maak het scherm schoon en ga naar (0,0):
      \033[2J
    - Verwijder tot het einde van de regel:
      \033[K

    - Bewaar de cursorpositie:
      \033[s
    - Herstel de cursorpositie:
      \033[u
```

De laatste twee codes worden NIET door veel terminal-emulators in acht genomen. De enige die ik ken zijn xterm en nxterm, zelfs al is de meerderheid van terminal-emulators gebaseerd op xterm-code. Zover ik weet, ondersteunen tell, rxvt, kvt, xiterm, en Eterm ze niet. Ze worden op de console ondersteund.

Probeer de volgende regel code achter de prompt te plaatsen (het wordt wat duidelijker als de prompt verscheidene regels lager ligt dan de bovenkant van de terminal, als je dit opgeeft):
`echo -en "\033[7A\033[1;35m BASH \033[7B\033[6D"`

Hiermee zou de cursor zeven regels naar boven moeten gaan, het woord "BASHäf moeten drukken en dan terugkeren naar waar het begon om een normale prompt te produceren. Dit is geen prompt, het is gewoon een demonstratie van de cursorbeweging op het scherm, waarbij kleur wordt gebruikt om te benadrukken wat er gebeurde.

Sla dit in een bestand op met de naam "clock":

```
#!/bin/bash

function prompt_command {
let prompt_x=$COLUMNS-5
```

```
}

PROMPT_COMMAND=prompt_command

function clock {
local       BLUE="\[\033[0;34m\]"
local        RED="\[\033[0;31m\]"
local  LIGHT_RED="\[\033[1;31m\]"
local      WHITE="\[\033[1;37m\]"
local  NO_COLOUR="\[\033[0m\]"
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="${TITLEBAR}\
\[\033[s\033[1;\$(echo -n \${prompt_x})H\]\
$BLUE[$LIGHT_RED\$(date +%H%M)$BLUE]\[\033[u\033[1A\]
$BLUE[$LIGHT_RED\u@\h:\w$BLUE]\
$WHITE\$$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

Deze prompt is tamelijk kaal, behalve dat het een 24-uurs klok in de rechterbovenhoek van de terminal bijhoudt (zelfs als de grootte van de terminal wordt gewijzigd). Dit zal NIET werken op de terminal-emulators die ik noemde die het opslaan en herstellen van de cursorpositie-codes niet accepteren. Als je deze prompt in één van deze terminal-emulators uitprobeert, zal de klok juist verschijnen, maar de prompt zal op de tweede regel van de terminal worden onderbroken.

Zie ook 12.7 (De Elegant Useless Clock Prompt) voor een wat uitgebreider gebruik van deze codes.

6.3 Verplaatsen van de Cursor met tput

Zoals met zoveel dingen onder Unix, zijn er vele manieren om hetzelfde te bereiken. Een utility genaamd "tput"kan ook worden gebruikt om de cursor op het scherm te manoeuvreren, of informatie terug te krijgen over de status van de terminal. "tputïs voor het positioneren van de cursor minder flexibel dan ANSI escape sequences. Je kan de cursor alleen naar een absolute positie verplaatsen. Je kunt het niet naar een positie verplaatsen die relatief is ten op zichte van de huidige positie. Ik maak geen gebruik van "tput", dus ik ga het je hier niet in detail uitleggen. Typ "man tputën dan zul je net zoveel weten als ik.

7 Speciale Tekens: Octale Escape Sequences

Buiten de tekens die je op je toetsenbord in kunt tikken, zijn er nog heel wat andere tekens die je op je scherm af kan drukken. Ik heb een script gemaakt waarmee het mogelijk is dat je kunt controleren wat het font dat je gebruikt, beschikbaar voor je heeft. Het belangrijkste commando dat je nodig hebt om deze tekens aan te passen is ëcho -e". De -e"switch vertelt echo de interpretatie van tekens voorafgegaan door een escape-teken, de backslash, te activeren. Wat je te zien krijgt wanneer je naar octaal 200-400 kijkt zal heel anders zijn met een VGA-font dan wat je te zien krijgt met een standaard Linux-font. Wees gewaarschuwd dat een aantal van deze escape sequences vreemde effecten hebben op je terminal, en ik heb het niet geprobeerd om datgene te voorkomen wat ze dan ook plegen te doen. De tekens voor het trekken van lijnen en tekenen van blokken (waar velen van ons bekend mee werden door WordPerfect) die veel door het Bashprompt project worden gebruikt, bevinden zich tussen octaal 260 en 337.

```
#!/bin/bash

#   Script: escgen

function usage {
   echo -e "\033[1;34mescgen\033[0m <lower_octal_value> [<higher_octal_value>]"
   echo " Octal escape sequence generator: druk alle octale escape sequences af"
   echo " tussen de lagere waarde en de hogere waarde. Als er geen tweede
        " waarde  wordt opgegeven, druk dan acht tekens af."
   echo " 1998 - Giles Orr, geen garantie."
   exit 1
}

if [ "$#" -eq "0" ]
then
   echo -e "\033[1;31mGeef alsjeblieft een of twee waarden op.\033[0m"
   usage
fi
let lower_val=${1}
if [ "$#" -eq "1" ]
then
   #   Als er geen afsluitende waarde wordt opgegeven, geef ze dan 8 tekens.
   upper_val=$(echo -e "obase=8 \n ibase=8 \n $lower_val+10 \n quit" | bc)
else
   let upper_val=${2}
fi
if [ "$#" -gt "2" ]
then
   echo -e "\033[1;31mGeef alsjeblieft twee waarden op.\033[0m"
   echo
   usage
fi
if [ "${lower_val}" -gt "${upper_val}" ]
then
   echo -e "\033[1;31m${lower_val} is groter dan ${upper_val}."
   echo
```

```
   usage
fi
if [ "${upper_val}" -gt "777" ]
   then
   echo -e "\033[1;31mWaarden kunnen de 777 niet overschrijden.\033[0m"
   echo
   usage
fi

let i=$lower_val
let line_count=1
let limit=$upper_val
while [ "$i" -lt "$limit" ]
do
   octal_escape="\\$i"
   echo -en "$i:'$octal_escape' "
   if [ "$line_count" -gt "7" ]
   then
      echo
      #   Voeg een harde return in.
      let line_count=0
   fi
   let i=$(echo -e "obase=8 \n ibase=8 \n $i+1 \n quit" | bc)
   let line_count=$line_count+1
done
echo
```

Je kunt ook xfd gebruiken om alle tekens van een X-font te tonen, met het commando, "xfd -fn <fontname>". Het klikken op een gegeven teken zal je heel veel informatie, waaronder de octale waarde, over dat teken geven. Het gegeven script van hierboven zal nuttig zijn voor op de console en als je niet zeker bent van de naam van het huidige font.

8 Het Bash Prompt Package

8.1 Beschikbaarheid

Het Bash Prompt package is beschikbaar op *http://bash.current.nu*, en is het werk van verscheidene mensen, gecoördineerd door Rob Current (ala BadLandZ). Het package is een vroege beta, maar biedt een eenvoudige manier om gebruik te maken van meerdere prompts (of themes), wat het mogelijk maakt dat je prompts voor login-shells, en voor subshells in kunt stellen (d.w.z. het plaatsen van PS1-strings in ~/.bash_profile en ~/.bashrc). De meeste themes maken gebruik van de uitgebreide VGA-character set, dus ze zien er slecht uit tenzij ze met VGA-fonts worden gebruikt (welke op de meeste systemen niet als standaard zijn ingesteld).

8.2 Xterm Fonts

Om een aantal van de meest attractieve prompts in het Bash Prompt package te gebruiken, moet je fonts ophalen en installeren die de character sets ondersteunen welke door de prompts worden verwacht. Dit zijn "VGA Fonts", welke verschillende character sets ondersteunen anders dan reguliere Xterm-fonts. Standaard Xterm fonts ondersteunen een uitgebreid alfabet, inclusief heel wat letters met accenten. In VGA fonts wordt dit materiaal vervangen door grafische tekens - blokken, punten, lijnen. Ik vroeg om een uitleg over dit verschil en Sérgio Vale e Pace (space@gold.com.br) schreef me:

Ik ben dol op computer-historie dus hier komt het:

Toen IBM de eerste PC ontwierp hadden ze een aantal te gebruiken character codes nodig, dus ze kregen de ASCII-character tabel (128 nummers, letters, en wat leestekens) en om een byte geadresseerde tabel te vullen, voegde ze nog 128 tekens toe. Sinds de PC werd ontworpen als home-computer, vulden ze de resterende 128 tekens met punten, lijnen, blokken, enz. om kaders en grijstint effecten te kunnen produceren (denk eraan dat we het hier hebben over 2 kleuren graphics).

Tijd gaat voorbij. PC's worden een standaard, IBM maakt krachtiger systemen en de VGA-standaard wordt geboren, samen met 256 kleuren graphics, en IBM gaat verder om hun IBM-ASCII character tabel op te nemen.

Meer tijd gaat voorbij. IBM heeft zijn leiderschap in de PC-markt verloren en de OS-auteurs ontdekken dat er andere talen in de wereld zijn die gebruik maken van niet-engelse tekens, dus voegen ze internationale ondersteuning toe aan hun systemen. Aangezien we nu intelligente kleurenschermen hebben, kunnen we de punten, lijnen, enz weggooien en die ruimte gebruiken voor geaccentueerde tekens en een aantal Griekse letters, die je in Linux zult zien.

8.3 Wijzigen van het Xterm Fontbdftopcf

Het verkrijgen en installeren van deze fonts is een wat ingewikkeld proces. Haal als eerste de fonts op. Verzeker jezelf er vervolgens van dat het .pcf of .pcf.gz bestanden zijn. Als het .bdf bestanden zijn onderzoek dan het "bdftopcf"commando (d.w.z. lees de man page). Plaats de .pcf of .pcf.gz bestanden in de /usr/X11R6/lib/X11/fonts/misc dir (dit is de juiste directory voor RedHat 5.1 en Slackware 3.4. Het kan voor andere distributies anders zijn). "cd"naar die directory en start het "mkfontdir"commando op. Start dan "xset fp rehashöp. Soms is het een goed idee het fonts.alias bestand in dezelfde directory te openen en kortere aliasnamen voor de fonts aan te maken.

Om de nieuwe fonts te gebruiken, start je het Xterm-programma naar keuze met het juiste commando voor je Xterm. Dit kan in de man page worden gevonden of met de --help"parameter op de commandoregel. Populaire terms zouden als volgt kunnen worden gebruikt:

```
xterm -font <fontname>
```

OF

```
xterm -fn <fontname> -fb <fontname-bold>
Eterm -F <fontname>
rxvt -fn <fontname>
```

VGA fonts zijn beschikbaar vanaf *Stumpy's ANSI Fonts* page bij *http://home.earthlink.net/˜us5zahns/enl/ansifont.html* (waarvan ik veel heb gebruik gemaakt tijdens dit schrijven).

9 Laden van een Andere Prompt

9.1 Laden van een Andere Prompt, Later

In deze HOWTO is aangegeven hoe PS1-omgevingsvariabelen kunnen worden aangemaakt, of hoe die PS1- en PS2-strings in functies kunnen worden opgenomen, welke door ~/.bashrc of als een theme door het bashprompt package kunnen worden aangeroepen.

Met het bashprompt package, kun je `bashprompt -i` intikken om een lijst met beschikbare themes te zien te krijgen. Voor het instellen van de prompt in toekomstige login-shells (in eerste instantie de console, maar ook telnet en Xterms, afhankelijk van hoe je Xterms zijn ingesteld), tik je in `bashprompt -l themename`. Bashprompt wijzigt dan je ~/.bash_profile om het verzochte theme aan te roepen wanneer het start. Het instellen van de prompt in toekomstige subshells (meestal Xterms, rxvt, enz.), gaat door het intikken van `bashprompt -s themename`, en bashprompt wijzigt je bestand ~/.bashrc om het juiste theme tijdens het opstarten aan te roepen.

Zie ook 2.6 (Permanent Instellen van de PS?-Strings) voor de notitie van Johan Kullstam betreft het belang van het plaatsen van de PS?-strings in ~/.bashrc.

9.2 Laden van een Andere Prompt, Onmiddellijk

Je kunt de prompt in je huidige terminal veranderen (met behulp van de voorbeeldfunctie ëlite"van hiervoor) door het typen van `"source elite"gevolgd door "elite"(in de veronderstelling dat de elite functie in de werkdirectory voorkomt). Dit is wat omslachtig en het laat je achter met een extra functie (elite) in de ruimte van je omgeving. Als je de omgeving op wilt schonen, zou je bovendien "unset elite moeten typen. Dit ziet er uit als een ideale kandidaat voor een klein shell-script, maar een script werkt hier niet omdat het script de omgeving van je huidige shell niet kan wijzigen: het kan alleen de omgeving van de subshell waaronder het draait wijzigen. Zodra het script stopt, verdwijnt de subshell, en wijzigingen die het script aan de omgeving had toegebracht, zijn er niet meer. Wat de omgevingsvariabelen van je huidige shell wel kan veranderen zijn omgevingsfuncties. Het bashprompt package plaatst een functie genaamd "callbashpromptïn je omgeving en ondanks dat ze het niet documenteren, kan het worden geladen om iedere bashprompt theme tijdens het werken te laden. Het kijkt in de theme directory dat het installeerde (het theme dat je aanriep moet zich hier bevinden), past een source toe op de functie waarnaar je vroeg, laadt de functie, en past dan een unset toe op de functie, dus je omgeving zonder rommel achterlatend. Het was niet de bedoeling dat "callbashpromptöp deze manier gebruikt zou worden en het bevat geen controle op fouten, maar als je dat in gedachten houdt, werkt het tamelijk goed.`

10 Dynamisch Laden van Prompt Kleuren

10.1 Een "Proof of Concept"Voorbeeld

`Dit is meer een "proof of concept"dan een attractieve prompt: het dynamisch wijzigen van de kleuren in een prompt. In het voorbeeld wijzigt de kleur van de hostnaam afhankelijk van de load (als een waarschuwing).`

```
#!/bin/bash
```

```
#   "hostloadcolour" - 17 October 98, door Giles
#
#   De gedachte hier is de kleur van de hostnaam in de prompt te wijzigen,
#   afhankelijk van de grens van een load waarde.

# THRESHOLD_LOAD is de waarde van de een minuut load (vermenigvuldigd
# met honderd) waarop je wilt dat de prompt
# wijzigt van COLOUR_LOW naar COLOUR_HIGH
THRESHOLD_LOAD=200
COLOUR_LOW='1;34'
          # lichtblauw
COLOUR_HIGH='1;31'
           # lichtrood

function prompt_command {
ONE=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
#   Apparently, "scale" in bc doesn't apply to multiplication, but does
#   apply to division.
ONEHUNDRED=$(echo -e "scale=0 \n $ONE/0.01 \nquit \n" | bc)
if [ $ONEHUNDRED -gt $THRESHOLD_LOAD ]
then
    HOST_COLOUR=$COLOUR_HIGH
        # Lichtrood
else
    HOST_COLOUR=$COLOUR_LOW
        # Lichtblauw
fi
}

function hostloadcolour {

PROMPT_COMMAND=prompt_command
PS1="[$(date +%H%M)][\u@\[\033[\$(echo -n \$HOST_COLOUR)m\]\h\[\033[0m\]:\w]$ "
}
```

Sla dit met je favoriete editor op in een bestand met de naam "hostloadcolour". Als je het Bashprompt package hebt geïnstalleerd, zal dit als een theme werken. Typ source hostloadcolour en vervolgens hostloadcolour als dit niet zo is. In beide gevallen wordt "prompt_commandëen functie in je omgeving. Als je de code bestudeert, zul je opmerken dat de kleuren ($COLOUR_HIGH en $COLOUR_LOW) zijn ingesteld door slechts gebruik te maken van een gedeeltelijke kleurcode, d.w.z. "1;34ïn plaats van "\[\033[1;34m\]", waar ik de voorkeur aan zou hebben gegeven. Het lukte me niet om het met de complete code aan het werk te krijgen. Laat het me alsjeblieft weten als je het voor elkaar krijgt.

11 Prompt Code Fragmenten

Deze sectie toont hoe diverse stukjes informatie in de Bash-prompt kunnen worden gezet. Er zijn oneindig veel dingen die in je prompt gezet zouden kunnen worden. Stuur me gerust wat voorbeelden op. Ik zal datgene waarvan ik denk dat ze het meest zullen worden gebruikt, proberen in te voegen. Als je een alternatieve manier hebt om wat van de informatie hier, op te halen, en je het gevoel hebt dat jouw methode efficiënter is, neem dan alsjeblieft contact met me op. Het is heel makkelijk slechte code te schrijven. Ik doe dit vaak, maar het is geweldig om elegante code te schrijven, en een plezier het te lezen. Ik krijg het zo af en toe voor elkaar en zou het bijzonder prettig vinden er hiervan meer in te plaatsen.

Om shell-code in prompts op te nemen, moet er gebruik worden gemaakt van escape-tekens. Meestal betekent dit dat het tussen \$(<commando>) moet worden geplaatst, zodat de uitvoer van het commando iedere keer dat de prompt wordt gegenereerd, wordt gesubstitueerd.

11.1 Ingebouwde Escape Sequences

Zie 2.5 (Bash Prompt Escape Sequences) voor een volledige lijst met ingebouwde escape sequences. Deze lijst is rechtsstreeks afkomstig uit de man page van Bash, dus je zou ook daarin kunnen kijken.

11.2 Datum en Tijd

Mocht je de interne commando's date en time niet prettig vinden, het extraheren van de informatie vanuit het date commando is relatief eenvoudig. Voorbeelden die reeds in deze HOWTO te zien waren, zijn date +%H%M, die het uur in 24-uurs formaat zet, en de minuut. date "+%A, %d %B %Y"

geeft je iets als "Sunday, 06 June 1999". Typ date -help of man date voor een volledige lijst met geïnterpreteerde reeksen.

11.3 Tellen van Bestanden in de Huidige Directory

Om vast te stellen hoeveel bestanden er zich in de huidige directory bevinden, plaats je er ls -l | wc -l in. Dit maakt gebruik van wc (wordcount) om het aantal regels (-l) van de uitvoer van ls -l te tellen. Bestanden die met een punt beginnen worden niet meegeteld. Als je alleen de bestanden wilt meetellen en de symbolische links hierin NIET op wilt nemen (gewoon een voorbeeld hoe je het anders zou kunnen doen), zou je gebruik kunnen maken van ls -l | grep -v ^l | wc -l. Hier controleert grep iedere regel op een beginnende "l"(wat een link aangeeft) en verwerpt die regel (-v).

11.4 Totaal aantal Bytes in de Huidige Directory

Als je wilt weten hoeveel ruimte de inhoud van de huidige directory in beslag neemt, kun je iets als wat volgt gebruiken:

```
# Het commando sed vervangt alle spaties door slechts één spatie.
# cut -d" " -f5 : -d stelt een scheidingsteken vast, wat betekent dat (in dit
# geval) een spatie een nieuwe kolom begint.
# -f geeft aan dat een bepaalde kolom eruit moet worden genomen, wat in dit
# geval de vijfde kolom is

let TotalBytes=0
```

```
for Bytes in $(ls -l | grep "^-" | sed -e "s/ \+/ /g" | cut -d" " -f5)
do
   let TotalBytes=$TotalBytes+$Bytes
done

# De if...fi's geeft een specifieker uitvoer in byte, kilobyte, megabyte,
# en gigabyte

if [ $TotalBytes -lt 1024 ]; then
   TotalSize=$(echo -e "scale=3 \n$TotalBytes \nquit" | bc)
else if [ $TotalBytes -lt 1048576 ]; then
   TotalSize=$(echo -e "scale=3 \n$TotalBytes/1024 \nquit" | bc)
else if [ $TotalBytes -lt 1073741824 ]; then
   TotalSize=$(echo -e "scale=3 \n$TotalBytes/1048576 \nquit" | bc)
else
   TotalSize=$(echo -e "scale=3 \n$TotalBytes/1073741824 \nquit" | bc)
fi
fi
fi
```

De code dankzij de welwillendheid van Sam Schmit (id@pt.lu) en zijn oom Jean-Paul, die een tamelijk belangrijke bug in mijn originele code glad streken, en het in het algemeen opschoonden.

11.5 Controleren op Huidige TTY

Het tty commando retourneert de bestandsnaam van de terminal die is verbonden met de standaardinvoer. Dit is er in twee formaten op de Linux systemen die ik heb gebruikt, óf /dev/tty4"óf /dev/pts/2". Ik ben hier een meer algemene oplossing voor gaan gebruiken: tty | sed -e "s:/dev/::"

, waarmee de voorafgaande /dev/"wordt verwijderd. Oudere systemen (in mijn ervaring, tot aan RedHat 5.2) retourneerden alleen bestandnamen in het formaat /dev/tty4", dus gebruikte ik tty | sed -e "s/.*tty\(.*\)/\1/"

.

Een alternatieve methode: ps aux | grep $$ | awk '{ print $7 }'.

11.6 Uitgestelde Job Telling

Om erachter te komen hoeveel uitgestelde taken je hebt, gebruik je jobs | wc -l | awk '{print $1}'. Overigens rekent awk spaties mee wat ruimte in een prompt verspilt. Als je netscape vanuit een xterm opstart, zal dat ook worden geteld. Als je dat wilt voorkomen en alleen gestopte jobs wilt tellen, gebruik je in plaats daarvan jobs -s. Typ help jobs voor meer informatie over jobs. jobs zal in Bash versie 2.02 nooit iets naar een pipe retourneren. Dit probleem is in geen enkele andere versie aanwezig.

11.7 Uptime en Load

Huidige load is afkomstig van het uptime commando. Wat ik thans gebruik is uptime | sed -e "s/.*load average: \(.*\...\), .*\..., .*\.../\1/-e "s/ //g" wat extreem onhandig is, maar het werkt. Verbeteringen zijn welkom. uptime kan ook op vergelijkbare manier worden gebruikt om er achter te komen hoe lang de computer werkend is (vanzelfsprekend) en hoeveel gebruikers zijn ingelogd. De gegevens zouden kunnen worden verwerkt met sed om ze er uit te laten zien zoals je dat wilt.

11.8 Aantal Processen

ps ax | wc -l | tr -d *OF* ps ax | wc -l | awk '{print $1}' *OF* ps ax | wc -l | sed -e "s: ::g"

. In ieder geval, wordt tr of awk of sed gebruikt om de ongewenste spaties te verwijderen.

11.9 Beheren van de Breedte van $PWD

Unix staat lange bestandsnamen toe, wat er toe kan leiden dat de waarde van $PWD nogal lang wordt. Een aantal mensen (met name de standaard RedHat prompt) kozen ervoor de basename van de huidige werkdirectory te gebruiken (bv. "gilesäls $PWD=/home/giles"). Ik wil graag wat meer informatie dan dat, maar het is vaak wenselijk de lengte van de directory binnen de perken te houden, en het is het meest zinvol aan de linkerkant af te kappen.

```
#   Hoeveel tekens van $PWD moeten bewaard blijven
local pwd_length=30
if [ $(echo -n $PWD | wc -c | tr -d " ") -gt $pwd_length ]
then
   newPWD="...$(echo -n $PWD | sed -e "s/.*\(.\{$pwd_length\}\)/\1/")"
else
   newPWD="$(echo -n $PWD)"
fi
```

De code hierboven kan worden uitgevoerd als onderdeel van PROMPT_COMMAND, en de gegenereerde omgevingsvariabele (*newPWD*) kan vervolgens in de prompt worden ingevoegd.

11.10 Laptop Power

Nogmaals, dit is niet elegant, maar het werkt (meestal). Als je een laptop hebt met daarop APM geïnstalleerd, probeer dan power=$(apm | sed -e "s/.*: \([1-9][0-9]*\)%/\1/"| tr -d) uitgevoerd vanaf PROMPT_COMMAND om een omgevingsvariabele aan te maken die je aan je prompt toe kunt voegen. Hiermee zal de resterende stroom worden weergegeven.

11.11 De Prompt Negeren bij Knippen en Plakken

Deze is vreemd maar gaaf. Rory Toma (rory@corp.webtv.net) schreef om een prompt als deze van de hand te doen : : rory@demon ; . Hoe is dit te gebruiken? Als je een commando achter de prompt typt (vreemd idee), dan kun je driemaal snel achtereen op die regel klikken (tenminste onder Linux) om de gehele regel op te lichten, en die regel dan voor een andere prompt plakken, en datgene tussen de ":ën de ":ën de "@"wordt als volgt genegeerd:

```
: rory@demon ; uptime
  5:15pm  up 6 days, 23:04,  2 users,  load average: 0.00, 0.00, 0.00
: rory@demon ; : rory@demon ; uptime
  5:15pm  up 6 days, 23:04,  2 users,  load average: 0.00, 0.00, 0.00
```

Als PS2 op een spatie is ingesteld, kunnen bovendien meerdere regels worden geknipt en geplakt.

11.12 Het Apart Instellen van de Venstertitel en Ikoon-titel

Een suggestie van Charles Lepple (clepple@negativezero.org) over het apart instellen van de venstertitel van Xterm en de titel van het overeenkomstige ikoon. (kijk eerst in de eerdere sectie 5 (Xterm Titelbalk Manipulaties)). Hij gebruikt dit onder WindowMaker omdat de titel die geschikt is voor een Xterm gewoonlijk te lang is voor een 64x64 ikoon. "\[\e]1;icon-title\007\e]2;main-title\007\]". Hij zegt dit in het prompt-commando in te stellen, omdat "Ik probeerde de string in PS1 te plaatsen, maar het veroorzaakte onder een aantal window managers geflikker omdat het resulteert in het meerdere malen instellen van de prompt wanneer je een uit meerdere regels bestaand commando gebruikt. (tenminste onder bash 1.4.x - en ik was te lui om volledig uit te zoeken wat de redenen hiervoor waren)."Ik had er in de PS1-string geen problemen mee, maar maakte geen gebruik van uit meerdere regels bestaande commando's. Hij wees me er ook op dat het onder xterm, xwsh, en dtterm werkt, maar niet met de gnome-terminal (welke slechts gebruik maakt van de hoofdtitel). Ik kwam erachter dat het ook met rxvt werkte, maar niet met kterm.

12 Voorbeeldprompts

12.1 Voorbeelden op het Web

In de loop van de tijd hebben veel mensen me uitstekende voorbeelden gemaild, en ik heb er zelf ook een aantal interessante geschreven. Er zijn er veel te veel om hier in op te nemen, dus ik heb alle voorbeelden bijelkaar op een webpage geplaatst die kan worden bekeken op *http://www.interlog.com/~giles/bashprompt/prompts*. Op Webpages kan ik ook afbeeldingen invoegen, wat ik in een standaard HOWTO niet kan doen. Alle voorbeelden die hierin staan, behalve die van Bradley Alexander "Prompts Depending on Connection Types"kunnen ook op het web worden bekeken.

12.2 Een "Lichtgewicht"Prompt

```
function proml {
local BLUE="\[\033[0;34m\]"
local RED="\[\033[0;31m\]"
local LIGHT_RED="\[\033[1;31m\]"
local WHITE="\[\033[1;37m\]"
local NO_COLOUR="\[\033[0m\]"
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
```

```
    *)
        TITLEBAR=""
        ;;
esac

PS1="${TITLEBAR}\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$LIGHT_RED\u@\h:\w$BLUE]\
$WHITE\$$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

12.3 Elite van Bashprompt Themes

Merk op dat hier een VGA-font voor is vereist.

```
# Aangemaakt door KrON van windowmaker op IRC
# Gewijzigd door Spidey 08/06
function elite {
PS1="\[\033[31m\]\332\304\[\033[34m\](\[\033[31m\]\u\[\033[34m\]@\[\033[31m\]\h\
\[\033[34m\])\[\033[31m\]-\[\033[34m\](\[\033[31m\]\$(date +%I:%M%P)\
\[\033[34m\]-:-\[\033[31m\]\$(date +%m)\[\033[34m\033[31m\]/\$(date +%d)\
\[\033[34m\])\[\033[31m\]\304-\[\033[34m]\\371\[\033[31m\]-\371\371\
\[\033[34m\]\372\n\[\033[31m\]\300\304\[\033[34m\](\[\033[31m\]\W\[\033[34m\])\
\[\033[31m\]\304\371\[\033[34m\]\372\[\033[00m\]"
PS2="> "
}
```

12.4 Een "Power User"Prompt

Ik maak echt gebruik van deze prompt, maar het resulteert in opmerkelijke onderbrekingen bij het verschijnen van de prompt op een single-user PII-400, dus ik zou je aan willen raden het echt op een multi-user P-100 of iets dergelijks te gebruiken ... Bekijk het om ideeën op te doen, in plaats van het als een praktische prompt te beschouwen.

```
#!/bin/bash
#----------------------------------------------------------------------
#       POWER USER PROMPT "pprom2"
#----------------------------------------------------------------------
#
#   Aangemaakt augustus 98, Laatst gewijzigd 9 november 98 door Giles
#
#   Probleem: als de load afneemt, geeft het aan "1.35down-.08", get rid
#   of the negative

function prompt_command
{
```

```
#   Creëer TotalMeg variabele: som van zichtbare bestandsgroottes in
#   huidige directory
local TotalBytes=0
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \nx=$TotalBytes/1048576\n if (x<1) {print \"0\"} \n print x \nquit" | bc)

#      Dit wordt gebruikt om het verschil in load waarden te berekenen
#      waarin voorzien door het "uptime" commando.  "uptime" geeft load
#      gemiddelden op 1, 5, en 15 minuten markeringen.
#
local one=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
local five=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\).*/\2/" -e "s/ //g")
local diff1_5=$(echo -e "scale = scale ($one) \nx=$one - $five\n if (x>0) {print \"up\"} else {print \"down\"}\n print x \nquit \n" | bc)
loaddiff="$(echo -n "${one}${diff1_5}")"

#   Tel zichtbare bestanden:
let files=$(ls -l | grep "^-" | wc -l | tr -d " ")
let hiddenfiles=$(ls -l -d .* | grep "^-" | wc -l | tr -d " ")
let executables=$(ls -l | grep ^-..x | wc -l | tr -d " ")
let directories=$(ls -l | grep "^d" | wc -l | tr -d " ")
let hiddendirectories=$(ls -l -d .* | grep "^d" | wc -l | tr -d " ")-2
let linktemp=$(ls -l | grep "^l" | wc -l | tr -d " ")
if [ "$linktemp" -eq "0" ]
then
    links=""
else
    links=" ${linktemp}l"
fi
unset linktemp
let devicetemp=$(ls -l | grep "^[bc]" | wc -l | tr -d " ")
if [ "$devicetemp" -eq "0" ]
then
    devices=""
else
    devices=" ${devicetemp}bc"
fi
unset devicetemp

}

PROMPT_COMMAND=prompt_command

function pprom2 {

local        BLUE="\[\033[0;34m\]"
local  LIGHT_GRAY="\[\033[0;37m\]"
local LIGHT_GREEN="\[\033[1;32m\]"
```

```
local  LIGHT_BLUE="\[\033[1;34m\]"
local  LIGHT_CYAN="\[\033[1;36m\]"
local      YELLOW="\[\033[1;33m\]"
local       WHITE="\[\033[1;37m\]"
local         RED="\[\033[0;31m\]"
local   NO_COLOUR="\[\033[0m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$RED\u@\h$BLUE]\
$BLUE[\
$LIGHT_GRAY\${files}.\${hiddenfiles}-\
$LIGHT_GREEN\${executables}x \
$LIGHT_GRAY(\${TotalMeg}Mb) \
$LIGHT_BLUE\${directories}.\
\${hiddendirectories}d\
$LIGHT_CYAN\${links}\
$YELLOW\${devices}\
$BLUE]\
$BLUE[${WHITE}\${loaddiff}$BLUE]\
$BLUE[\
$WHITE\$(ps ax | wc -l | sed -e \"s: ::g\")proc\
$BLUE]\
\n\
$BLUE[$RED\$PWD$BLUE]\
$WHITE\$\
\
$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

12.5 Prompt Afhankelijk van het Type Verbinding

Bradley M Alexander (storm@tux.org) had het uitstekende idee om zijn gebruikers er aan te herinneren van welke verbinding ze op zijn computer(s) gebruik maakte, dus codeerde hij prompts in een kleur afhankelijk van het type verbinding. Hier is het bashrc dat hij me leverde:

```
# /etc/bashrc
```

```
# Systeemomvattende functies en aliassen
# Omgevingszaken gaan in /etc/profile

# Om onbekende reden weigert bash in een aantal situaties waar ik niet
# uit kom, PS1 te erven.
# Als PS1 hierin wordt geplaatst, ben je er zeker van dat het iedere keer
# wordt geladen.

# Stel prompts in. Kleurcode voor logins. Rood voor root, wit voor
# gebruikerslogins, groen voor ssh-sessies, cyaan voor telnet,
# magenta met rood "(ssh)" voor ssh + su, magenta voor telnet.
THIS_TTY=tty‘ps aux | grep $$ | grep bash | awk ’{ print $7 }’‘
SESS_SRC=‘who | grep $THIS_TTY | awk ’{ print $6 }’‘

SSH_FLAG=0
SSH_IP=‘echo $SSH_CLIENT | awk ’{ print $1 }’‘
if [ $SSH_IP ] ; then
  SSH_FLAG=1
fi
SSH2_IP=‘echo $SSH2_CLIENT | awk ’{ print $1 }’‘
if [ $SSH2_IP ] ; then
  SSH_FLAG=1
fi
if [ $SSH_FLAG -eq 1 ] ; then
  CONN=ssh
elif [ -z $SESS_SRC ] ; then
  CONN=lcl
elif [ $SESS_SRC = "(:0.0)" -o $SESS_SRC = "" ] ; then
  CONN=lcl
else
  CONN=tel
fi

# Okay...Wie zijn we nu?
if [ ‘/usr/bin/whoami‘ = "root" ] ; then
  USR=priv
else
  USR=nopriv
fi

#Stel een aantal prompts in...
if [ $CONN = lcl -a $USR = nopriv ] ; then
  PS1="[\u \W]\\$ "
elif [ $CONN = lcl -a $USR = priv ] ; then
  PS1="\[\033[01;31m\][\w]\\$\[\033[00m\] "
elif [ $CONN = tel -a $USR = nopriv ] ; then
  PS1="\[\033[01;34m\][\u@\h \W]\\$\[\033[00m\] "
elif [ $CONN = tel -a $USR = priv ] ; then
  PS1="\[\033[01;30;45m\][\u@\h \W]\\$\[\033[00m\] "
elif [ $CONN = ssh -a $USR = nopriv ] ; then
```

```
  PS1="\[\033[01;32m\][\u@\h \W]\\$\[\033[00m\] "
elif [ $CONN = ssh -a $USR = priv ] ; then
  PS1="\[\033[01;35m\][\u@\h \W]\\$\[\033[00m\] "
fi

# PS1="[\u@\h \W]\\$ "
export PS1
alias which="type -path"
alias dir="ls -lF --color"
alias dirs="ls -lFS --color"
alias h=history
```

12.6 Een Prompt ter Breedte van Je Term

Een vriend klaagde dat hij een prompt, die qua lengte bleef wijzigen, dit omdat $PWD er in voorkwam, niet prettig vond. Ik schreef dus een prompt die de grootte exact aan de breedte van je term wijzigde, met de werkdirectory in de bovenste regel van twee regels.

```
#!/bin/bash

#   termbrede prompt
#      door Giles - aangemaakt 2 November 98
#
#   De bedoeling hier is dat de bovenste van de uit twee regels bestaande
#   prompt altijd even breed is als de breedte van je term. Doe dit door
#   de breedte van de tekstelementen te berekenen, en vul het zonodig
#   uit of kap $PWD aan de linkerkant af.
#

function prompt_command {

TERMWIDTH=${COLUMNS}

#   Bereken de breedte van de prompt:

hostnam=$(echo -n $HOSTNAME | sed -e "s/[\.].*//")
#   "whoami" en "pwd" sluiten een afsluitende newline in
usernam=$(whoami)
let usersize=$(echo -n $usernam | wc -c | tr -d " ")
newPWD="${PWD}"
let pwdsize=$(echo -n ${newPWD} | wc -c | tr -d " ")
#   Voeg alle accessoires hieronder toe ...
let promptsize=$(echo -n "--(${usernam}@${hostnam})---(${PWD})--" \
                 | wc -c | tr -d " ")
let fillsize=${TERMWIDTH}-${promptsize}
fill=""
while [ "$fillsize" -gt "0" ]
do
   fill="${fill}-"
```

```
   let fillsize=${fillsize}-1
done

if [ "$fillsize" -lt "0" ]
then
   let cut=3-${fillsize}
   newPWD="...$(echo -n $PWD | sed -e "s/\(^.\{$cut\}\)\(.*\)/\2/")"
fi
}

PROMPT_COMMAND=prompt_command

function termwide {

local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local WHITE="\[\033[1;37m\]"
local NO_COLOUR="\[\033[0m\]"

local LIGHT_BLUE="\[\033[1;34m\]"
local YELLOW="\[\033[1;33m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\${usernam}$LIGHT_BLUE@$YELLOW\${hostnam}\
${LIGHT_BLUE})-${YELLOW}-\${fill}${LIGHT_BLUE}-(\
$YELLOW\${newPWD}\
$LIGHT_BLUE)-$YELLOW-\
\n\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\$(date +%H%M)$LIGHT_BLUE:$YELLOW\$(date \"+%a,%d %b %y\")\
$LIGHT_BLUE:$WHITE\$$LIGHT_BLUE)-\
$YELLOW-\
$NO_COLOUR "

PS2="$LIGHT_BLUE-$YELLOW-$YELLOW-$NO_COLOUR "

}
```

12.7 De Elegant Useless Clock Prompt

Dit is één van de meer aantrekkelijke (en nutteloze) prompts die ik heb gemaakt. Omdat veel van de X-terminal emulators het bewaren en herstellen van de positie van de cursor niet implementeren, kun je als alternatief de cursor onderaan de terminal verankeren als je een klok in de bovenste rechterhoek plaatst. Dit bouwt voort op het idee van de "termbrede"prompt van hiervoor, een lijn tekenend aan de rechterkant van het scherm van de prompt tot aan de klok. Een VGA-font is vereist.

Noot: Er is hier een vreemde substitutie, waardoor het waarschijnlijk niet juist wordt afgedrukt, als het vanuit SGML naar andere formaten is omgezet. Ik moest het schermteken voor \304 substitueren. Ik zou normaal gesproken gewoon de reeks "\304"hebben opgenomen, maar het was nodig in dit geval deze subsitutie te maken.

```
#!/bin/bash

#   Voor deze prompt is een VGA-font vereist. De prompt wordt aan de
#   onderkant van de terminal verankerd, vult op ter breedte van de terminal
#   en tekent een verticale lijn aan de rechterkant van de terminal
#   waarbij het zichzelf koppelt aan een klok in de rechterbovenhoek van
#   de terminal.

function prompt_command {
#   Bereken de breedte van de prompt:
hostnam=$(echo -n $HOSTNAME | sed -e "s/[\.].*//")
#   "whoami" en "pwd" nemen een afsluitende newline op
usernam=$(whoami)
newPWD="${PWD}"
#   Voeg alle accesoires hieronder in ...
let promptsize=$(echo -n "--(${usernam}@${hostnam})---(${PWD})-----" \
                 | wc -c | tr -d " ")
#   Zoek uit hoeveel er tussen user@host en PWD moet worden ingevoegd (of
#   hoeveel van PWD te verwijderen)
let fillsize=${COLUMNS}-${promptsize}
fill=""
#   Maak de opvuller als de prompt niet zo breed is als de terminal:
while [ "$fillsize" -gt "0" ]
do
   fill="${fill}Ä"
   # De A met de umlaut erboven (het verschijnt als een lange streep als
   # je een VGA-font gebruikt) is \304, maar ik knipte het en plakte het in
   # omdat Bash slechts één substitutie uitvoert en dat is
   # in dit geval het plaatsen van $fill in de prompt.
   let fillsize=${fillsize}-1
done
#   Kap PWD aan de rechterkant af als de prompt breder wordt dan de terminal:
if [ "$fillsize" -lt "0" ]
then
   let cutt=3-${fillsize}
   newPWD="...$(echo -n $PWD | sed -e "s/\(^.\{$cutt\}\)\(.*\)/\2/")"
fi
```

```
#
#   Maak de klok en de balk die aan de rechterkant van de term draait
#
local LIGHT_BLUE="\033[1;34m"
local     YELLOW="\033[1;33m"
#   Positioneer de cursor om de klok weer te geven:
echo -en "\033[2;$((${COLUMNS}-9))H"
echo -en "$LIGHT_BLUE($YELLOW$(date +%H%M)$LIGHT_BLUE)\304$YELLOW\304\304\277"
local i=${LINES}
echo -en "\033[2;${COLUMNS}H"
#  Print verticale streepjes:
while [ $i -ge 4 ]
do
   echo -en "\033[$(($i-1));${COLUMNS}H\263"
   let i=$i-1
done

let prompt_line=${LINES}-1
#   Dit is nodig omdat een \${LINES} binnen een Bash mathematische
#   expressie (d.w.z. $(())) niet schijnt te werken.
}

PROMPT_COMMAND=prompt_command

function clock3 {
local LIGHT_BLUE="\[\033[1;34m\]"
local     YELLOW="\[\033[1;33m\]"
local      WHITE="\[\033[1;37m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local  NO_COLOUR="\[\033[0m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
\[\033[\${prompt_line};0H\]
$YELLOW\332$LIGHT_BLUE\304(\
$YELLOW\${usernam}$LIGHT_BLUE@$YELLOW\${hostnam}\
${LIGHT_BLUE})\304${YELLOW}\304\${fill}${LIGHT_BLUE}\304(\
$YELLOW\${newPWD}\
$LIGHT_BLUE)\304$YELLOW\304\304\304\331\
\n\
$YELLOW\300$LIGHT_BLUE\304(\
$YELLOW\$(date \"+%a,%d %b %y\")\
```

```
$LIGHT_BLUE:$WHITE\$$LIGHT_BLUE)\304\
$YELLOW\304\
$LIGHT_GRAY "

PS2="$LIGHT_BLUE\304$YELLOW\304$YELLOW\304$NO_COLOUR "

}
```